REPORT DOCUMENTATION PAGE

Form Approved
OPM No. 0704-0188

1 hour per response, including the time for reviewing instructions, searching existing data sources gathering and comments regarding this burden estimate or any other aspect of this collection of information, including suggestions Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to st, Washington, DC 20503.

AD-A226 796

1. AG	DATE	3. REPORT TYPE AND DATES COVERED
		Final 13 Feb. 1990 to 13 Feb. 1991

4. TITLE AND SUBTITLE Ada Compiler Validation Summary Report:TLD Systems Ltd., TLD HP/MIL-STD-1750A Ada Compiler System, Version 2.1.3 Hewlett Packard HP 9000/350 (Host) to Hewlett Packard HP64000 with Tasco MDC281 Emulator (Target), 900213W1.10247

5. FUNDING NUMBERS

6. AUTHOR(S)

Wright-Patterson AFB, Dayton, OH
USA

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Ada Validation Facility, Language Control Facility ASD/SCEL
Bldg. 676, Rm 135
Wright-Patterson AFB
Dayton, OH 45433

8. PERFORMING ORGANIZATION REPORT NUMBER

AVF-VSR-359.0790

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Ada Joint Program Office
United States Department of Defense
Washington, D.C. 20301-3081

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

TLD Systems, Ltd., TLD HP/MIL-STD-1750A Ada Compiler System, Version 2.1.3, Wright-Patterson AFB, Hewlett Packard HP 9000/350 (Host) to Hewlett Packard HP64000 with Tasco MDC281 Emulator (Target), ACVC 1.10.

DTIC
ELECTE
SEP 25 1990
D

14. SUBJECT TERMS Ada programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, Validation Testing, Ada Validation Office, Ada Validation Facility, ANSI/MIL-STD-1815A, Ada Joint Program Office

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	

NSN 7540-01-280-5500

Standard Form 298. (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-01

90

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 900213W1.10247
TLD Systems, Ltd.
TLD HP/MIL-STD-1750A Ada Compiler System, Version 2.1.3
Hewlett Packard HP 9000/350 Host
and Hewlett Packard HP64000 with Tasco MDC281 Emulator Target

Completion of On-Site Testing:
13 February 1990

Prepared By:
Ada Validation Facility
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington DC 20301-3081

Accesion For

NTIS CRA&I		☑
DTIC TAB		☐
Unannounced		☐
Justification		

By

Distribution /

Availability Codes

Dist	Avail and / or Special
A-1	

Ada   Compiler Validation Summary Report:

Compiler Name: TLD HP/MIL-STD-1750A Ada Compiler System, Version 2.1.3

Certificate Number: 900213W1.10247

Host: Hewlett Packard HP 9000/350 under
          HP-UX, Version 4.0

Target: Hewlett Packard HP64000 with Tasco MDC281 Emulator under
          TLDspk Single Program Kernel, Version 2.1.3
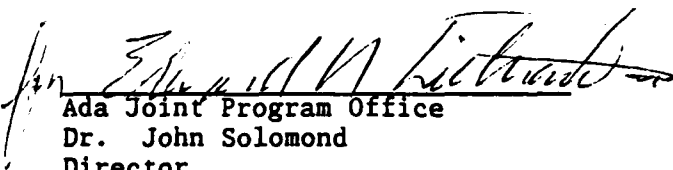

Testing Completed 13 February 1990 Using ACVC 1.10

Customer Agreement Number: 89-09-21-TLD

This report has been reviewed and is approved.

Ada Validation Facility
Steven P. Wilson
Technical Director
ASD/SCEL
Wright-Patterson AFB OH   45433-6503

Ada Validation Organization
Director, Computer & Software Engineering Division
Institute for Defense Analyses
Alexandria VA   22311

Ada Joint Program Office
Dr. John Solomond
Director
Department of Defense
Washington DC   20301

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation-dependent but is permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard

- To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard

- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by SofTech, Inc. under the direction of the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 13 February 1990 at Torrance CA.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C.#552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

> Ada Information Clearinghouse
> Ada Joint Program Office
> OUSDRE
> The Pentagon, Rm 3D-139 (Fern Street)
> Washington DC 20301-3081

or from:

> Ada Validation Facility
> ASD/SCEL
> Wright-Patterson AFB OH 45433-6503

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

> Ada Validation Organization
> Institute for Defense Analyses
> 1801 North Beauregard Street
> Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language,
   ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

2. Ada Compiler Validation Procedures, Version 2.0, Ada Joint Program
   Office, May 1989.

3. Ada Compiler Validation Capability Implementers' Guide, SofTech,
   Inc., December 1986.

4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

ACVC   The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.

Ada Commentary   An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.

Ada Standard   ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

Applicant   The agency requesting validation.

AVF   The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the Ada Compiler Validation Procedures.

AVO   The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.

Compiler   A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.

Failed test   An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.

Host   The computer on which the compiler resides.

Inapplicable   An ACVC test that uses features of the language that a
test   compiler is not required to support or may legitimately support in a way other than the one expected by the test.

Passed test   An ACVC test for which a compiler generates the expected result.

Target   The computer for which a compiler generates code.

Test   A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.

Withdrawn   An ACVC test found to be incorrect and not used to check
test   conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce compilation or link errors because of the way in which a program library is used at link time.

Class A tests ensure the successful compilation of legal Ada programs with certain language constructs which cannot be verified at compile time. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check the run time system to ensure that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Class E tests are expected to execute successfully and check implementation-dependent options and resolutions of ambiguities in the Ada Standard. Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated. In some cases, an implementation may legitimately detect errors during compilation of the test.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of each test in the ACVC follows conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be

customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: TLD HP/MIL-STD-1750A Ada Compiler System, Version 2.1.3

ACVC Version: 1.10

Certificate Number: 900213W1.10247

Host Computer:

Machine:	Hewlett Packard HP 9000/350
Operating System:	HP-UX Version 4.0
Memory Size:	16 Megabytes

Target Computer:

Machine:	Hewlett Packard HP64000 with Tasco MDC281 Emulator
Operating System:	TLDspk Single Program Kernel Version 2.1.3
Memory Size:	64 Kilobytes

Communications Network: Internal HP Bus

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

   a. Capacities.

      (1) The compiler correctly processes a compilation containing 723 variables in the same declarative part. (See test D29002K.)

      (2) The compiler correctly processes tests containing loop statements nested to 65 levels. (See tests D55A03A..H (8 tests).)

      (3) The compiler correctly processes tests containing block statements nested to 65 levels. (See test D56001B.)

      (4) The compiler correctly processes tests containing recursive procedures separately compiled as subunits nested to six levels. (See tests D64005E..G (3 tests).)

   b. Predefined types.

      (1) This implementation supports the additional predefined types LONG_INTEGER and LONG_FLOAT in package STANDARD. (See tests B86001T..Z (7 tests).)

   c. Expression evaluation.

      The order in which expressions are evaluated and the time at which constraints are checked are not defined by the language. While the ACVC tests do not specifically attempt to determine the order of evaluation of expressions, test results indicate the following:

      (1) Some of the default initialization expressions for record components are evaluated before any value is checked for membership in a component's subtype. (See test C32117A.)

      (2) Assignments for subtypes are performed with less precision than the base type. (See test C35712B.)

(3) This implementation uses no extra bits for extra precision and uses all extra bits for extra range. (See test C35903A.)

(4) Sometimes NUMERIC_ERROR is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

(5) No exception is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

(6) Underflow is gradual. (See tests C45524A..Z (26 tests).)

d. Rounding.

The method by which values are rounded in type conversions is not defined by the language. While the ACVC tests do not specifically attempt to determine the method of rounding, the test results indicate the following:

(1) The method used for rounding to integer is round away from zero. (See tests C46012A..Z (26 tests).)

(2) The method used for rounding to longest integer is round away from zero. (See tests C46012A..Z (26 tests).)

(3) The method used for rounding to integer in static universal real expressions is round away from zero. (See test C4A014A.)

e. Array types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT.

For this implementation:

(1) Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises NUMERIC_ERROR sometimes. (See test C36003A.)

(2) CONSTRAINT_ERROR is raised when 'LENGTH is applied to an array type with INTEGER'LAST + 2 components with each component being a null array. (See test C36202A.)

(3) CONSTRAINT_ERROR is raised when 'LENGTH is applied to an array type with SYSTEM.MAX_INT + 2 components with each component being a null array. (See test C36202B.)

2-3

(4) A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises CONSTRAINT_ERROR when the array objects are declared. (See test C52103X.)

(5) A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises CONSTRAINT_ERROR when the array objects are declared. (See test C52104Y.)

(6) A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises no exception. (See test E52103Y.)

(7) In assigning one-dimensional array types, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

(8) In assigning two-dimensional array types, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

f. Discriminated types.

(1) In assigning record types with discriminants, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

g. Aggregates.

(1) In the evaluation of a multi-dimensional aggregate, index subtype checks are made as choices are evaluated. (See tests C43207A and C43207B.)

(2) In the evaluation of an aggregate containing subaggregates, not all choices are evaluated before being checked for identical bounds. (See test E43212B.)

(3) CONSTRAINT_ERROR is raised before all choices are evaluated when a bound in a non-null range of a non-null aggregate does not belong to an index subtype. (See test E43211B.)

h. Pragmas.

(1) The pragma INLINE is not supported for functions or procedures. (See tests LA3004A..B (2 tests), EA3004C..D (2 tests), and CA3004E..F (2 tests).)

i. Generics.

(1) Generic specifications and bodies can be compiled in separate compilations. (See tests CA1012A, CA2009C, CA2009F, BC3204C, and BC3205D.)

(2) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

j. Input and output.

(1) The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

(2) The package DIRECT_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)

(3) The Director, AJPO, has determined (AI-00332) that every call to OPEN and CREATE must raise USE_ERROR or NAME_ERROR if file input/output is not supported. This implementation exhibits this behavior for SEQUENTIAL_IO, DIRECT_IO, and TEXT_IO.

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.10 of the ACVC comprises 3717 tests. When this compiler was tested, 44 tests had been withdrawn because of test errors. The AVF determined that 786 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 285 executable tests that use floating-point precision exceeding that supported by the implementation and 242 executable tests that use file operations not supported by the implementation. Modifications to the code, processing, or grading for 9 tests were required to successfully demonstrate the test objective.

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	127	1131	1560	15	10	44	2887
Inapplicable	2	7	755	2	18	2	786
Withdrawn	1	2	35	0	6	0	44
TOTAL	130	1140	2350	17	34	46	3717

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT						CHAPTER								TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	192	545	496	245	170	99	158	332	131	36	252	155	76	2887
Inappl	20	104	184	3	2	0	8	0	6	0	0	214	245	786
Wdrn	1	1	0	0	0	0	0	2	0	0	1	35	4	44
TOTAL	213	650	680	248	172	99	166	334	137	36	253	404	325	3717

3.4 WITHDRAWN TESTS

The following 44 tests were withdrawn from ACVC Version 1.10 at the time of this validation:

E28005C	A39005G	B97102E	C97116A	BC3009B	CD2A62D
CD2A63A	CD2A63B	CD2A63C	CD2A63D	CD2A66A	CD2A66B
CD2A66C	CD2A66D	CD2A73A	CD2A73B	CD2A73C	CD2A73D
CD2A76A	CD2A76B	CD2A76C	CD2A76D	CD2A81G	CD2A83G
CD2A84M	CD2A84N	CD2B15C	CD2D11B	CD5007B	CD50110
ED7004B	ED7005C	ED7005D	ED7006C	ED7006D	CD7105A
CD7203B	CD7204B	CD7205C	CD7205D	CE2107I	CE3111C
CE3301A	CE3411B				

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 786 tests were inapplicable for the reasons indicated:

a. The following 285 tests are not applicable because they have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

    C24113F..Y (20 tests)   C35705F..Y (20 tests)
    C35706F..Y (20 tests)   C35707F..Y (20 tests)
    C35708F..Y (20 tests)   C35802F..Z (21 tests)

```
C45241F..Y (20 tests)     C45321F..Y (20 tests)
C45421F..Y (20 tests)     C45521F..Z (21 tests)
C45524F..Z (21 tests)     C45621F..Z (21 tests)
C45641F..Y (20 tests)     C46012F..Z (21 tests)
```

b. C35702A and B86001T are not applicable because this implementation supports no predefined type SHORT_FLOAT.

c. The following 172 tests are not applicable because this implementation does not support 'SMALL clauses or 'SIZE length clauses for other than task types:

```
A39005B          A39005E          C87B62A          C87B62C
CD1009A..I (9)   CD1009L          CD10090..Q (3)   CD1C03A
CD1C03F          CD1C04A          CD1C04C          CD2A21A..E (5)
CD2A22A..J (10)  CD2A23A..E (5)   CD2A24A..J (10)  ED2A26A
CD2A31A..D (4)   CD2A32A..J (10)  CD2A41A..E (5)   CD2A42A..J (10)
CD2A51A..E (5)   CD2A52A..D (4)   CD2A52G..J (4)   CD2A53A..E (5)
CD2A54A..D (4)   CD2A54G..J (4)   ED2A56A          CD2A61A..L (12)
CD2A62A..C (3)   CD2A64A..D (4)   CD2A65A..D (4)   CD2A71A..D (4)
CD2A72A..D (4)   CD2A74A..D (4)   CD2A75A..D (4)   CD2A81A..F (6)
CD2A83A..C (3)   CD2A83E..F (2)   CD2A84B..I (8)   CD2A84K..L (2)
ED2A86A          CD2A87A          CD2D11A          CD2D13A
```

d. The following 16 tests are not applicable because this implementation does not support a predefined type SHORT_INTEGER:

```
C45231B   C45304B   C45502B   C45503B   C45504B
C45504E   C45611B   C45613B   C45614B   C45631B
C45632B   B52004E   C55B07B   B55B09D   B86001V
CD7101E
```

e. C45231D, B86001X, and CD7101G are not applicable because this implementation does not support any predefined integer type with a name other than INTEGER, LONG_INTEGER, or SHORT_INTEGER.

f. C45531M..P (4 tests) and C45532M..P (4 tests) are not applicable because the value of SYSTEM.MAX_MANTISSA is less than 48.

g. D64005F and D64005G are not applicable because this implementation does not support nesting 10 levels of recursive procedure calls.

h. C86001F is not applicable because, for this implementation, the package TEXT_IO is dependent upon package SYSTEM. This test recompiles package SYSTEM, making package TEXT_IO, and hence package REPORT, obsolete.

i. B86001Y is not applicable because this implementation supports no predefined fixed-point type other than DURATION.

j. B86001Z is not applicable because this implementation supports no predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT.

3-3

TEST INFORMATION

k. LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F are not applicable because this implementation does not support pragma INLINE.

l. The following 14 tests are not applicable because this implementation does not support record representation clauses:

CD1009N	CD1009X..Z (3)	CD1C03H	CD1C04E
ED1D04A	CD4031A	CD4041A	CD4051A..D (4)
CD7204C			

m. CD2B15B is not applicable to this implementation because the space that was reserved for the collection exceeded the specified size.

n. The following 29 tests are not applicable because this implementation does not support address clauses for constants:

CD5011B	CD5011D	CD5011F	CD5011H	CD5011L
CD5011N	CD5011R	CD5012C	CD5012D	CD5012G
CD5012H	CD5012L	CD5013B	CD5013D	CD5013F
CD5013H	CD5013L	CD5013N	CD5013R	CD5014B
CD5014D	CD5014F	CD5014H	CD5014J	CD5014L
CD5014N	CD5014R	CD5014U	CD5014W	

o. The following 242 tests are inapplicable because sequential, text, and direct access files are not supported:

CE2102A..C (3 tests)	CE2102G..H (2 tests)
CE2102K	CE2102N..Y (12 tests)
CE2103C..D (2 tests)	CE2104A..D (4 tests)
CE2105A..B (2 tests)	CE2106A..B (2 tests)
CE2107A..H (8 tests)	CE2107L
CE2108A..B (2 tests)	CE2108C..H (6 tests)
CE2109A..C (3 tests)	CE2110A..D (4 tests)
CE2111A..I (9 tests)	CE2115A..B (2 tests)
CE2201A..C (3 tests)	CE2201F..N (9 tests)
CE2204A..D (4 tests)	CE2205A
CE2208B	CE2401A..C (3 tests)
CE2401E..F (2 tests)	CE2401H..L (5 tests)
CE2404A..B (2 tests)	CE2405B
CE2406A	CE2407A..B (2 tests)
CE2408A..B (2 tests)	CE2409A..B (2 tests)
CE2410A..B (2 tests)	CE2411A
CE3102A..B (2 tests)	EE3102C
CE3102F..H (3 tests)	CE3102J..K (2 tests)
CE3103A	CE3104A..C (3 tests)
CE3107B	CE3108A..B (2 tests)
CE3109A	CE3110A
CE3111A..B (2 tests)	CE3111D..E (2 tests)
CE3112A..D (4 tests)	CE3114A..B (2 tests)
CE3115A	EE3203A

```
CE3208A              EE3301B
CE3302A              CE3305A
CE3402A              EE3402B
CE3402C..D (2 tests) CE3403A..C (3 tests)
CE3403E..F (2 tests) CE3404B..D (3 tests)
CE3405A              EE3405B
CE3405C..D (2 tests) CE3406A..D (4 tests)
CE3407A..C (3 tests) CE3408A..C (3 tests)
CE3409A              CE3409C..E (3 tests)
EE3409F              CE3410A
CE3410C..E (3 tests) EE3410F
CE3411A              CE3411C
CE3412A              EE3412C
CE3413A              CE3413C
CE3602A..D (4 tests) CE3603A
CE3604A..B (2 tests) CE3605A..E (5 tests)
CE3606A..B (2 tests) CE3704A..F (6 tests)
CE3704M..O (3 tests) CE3706D
CE3706F..G (2 tests) CE3804A..P (16 tests)
CE3805A..B (2 tests) CE3806A..B (2 tests)
CE3806D..E (2 tests) CE3806G..H (2 tests)
CE3905A..C (3 tests) CE3905L
CE3906A..C (3 tests) CE3906E..F (2 tests)

EE2201D..E (2 tests) EE2401D      EE2401G
```

p. The tests CE2103A, CE2103B, and CE3107A do not allow the implementation to raise USE_ERROR for external file CREATE and OPEN operations when these operations are not supported by the implementation. An unhandled USE_ERROR exception was raised by these tests.

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that wasn't anticipated by the test (such as raising one exception instead of another).

Modifications were required for 9 tests.

TEST INFORMATION

The following tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

    B24009A   B44004D   B49003A   B49005A   B49009C   B59001E
    BC3205A

Test AD7006A was modified because the Type SYSTEM.MEMORY_SIZE needed to be explicitly converted to type INTEGER. The modification was made so that line 23 was changed to:

    I := INTEGER (SYSTEM.MEMORY_SIZE - MYMSIZE + 1);

At the recommendation of the AVO, the expression "2**T'MANTISSA - 1" on line 262 of test CC1223A was changed to "(2**(T'MANTISSA-1)-1 + 2**(T'MANTISSA-1))" since the previous expression causes an unexpected exception to be raised.

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.10 produced by the TLD HP/MIL-STD-1750A Ada Compiler System, Version 2.1.3 compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the TLD HP/MIL-STD-1750A Ada Compiler System, Version 2.1.3 compiler using ACVC Version 1.10 was conducted on-site by a validation team from the AVF. The configuration in which the testing was performed is described by the following designations of hardware and software components:

Host computer:	Hewlett Packard HP 9000/350
Host operating system:	HP-UX, Version 4.0
Target computer:	Hewlett Packard HP64000
	with Tasco MDC281 Emulator
Target operating system:	TLDspk Single Program Kernel,
	Version 2.1.3
Compiler:	TLD HP/MIL-STD-1750A Ada Compiler
	System, Version 2.1.3

The host and target computers were linked via Internal HP Bus.

A magnetic tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precisions was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring modifications during the prevalidation testing were included in their modified form on the magnetic tape.

The contents of the magnetic tape were not loaded directly onto the host computer. The tests were loaded on the VAX, modified as necessary for dependencies, and transferred to the HP using Ethernet.

After the test files were loaded to disk, the full set of tests was compiled and linked on the Hewlett Packard HP 9000/350; then all executable images were transferred to the TLDsim MIL-STD-1750A Instruction Set Simulator via Internal HP Bus and run. Results were printed from the host computer.

The compiler was tested using command scripts provided by TLD Systems, Ltd. and reviewed by the validation team. The compiler was tested using the following option settings:

No_Exception Info   Suppress generation of Debug string's
                         in relocatable object code for unhandled
                         exception

No_Phase   Suppress displaying of phase times during
                         compilation

Model=MDC281   Selects the MDC281 model of the 1750A to
                         generate special sequences for this target

Tests were compiled, linked, and executed (as appropriate) using a single host and target computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

TEST INFORMATION

3.7.3 Test Site

Testing was conducted at Torrance CA and was completed on 13 February 1990.

APPENDIX A

DECLARATION OF CONFORMANCE

TLD Systems, Ltd. has submitted the following
Declaration of Conformance concerning the TLD
HP/MIL-STD-1750A Ada Compiler System, Version 2.1.3
compiler.

DECLARATION OF CONFORMANCE

Compiler Implementor: TLD Systems, Ltd.
Ada Validation Facility: ASD/SCOL, Wright-Patterson AFB, OH 45433-6503
Ada Compiler Validation Capability (ACVC) Version: 1.10

Base Configuration

Base Compiler Name: TLD HP/MIL-STD-1750A Ada Compiler System, Version 2.1.3

Host Architecture ISA: Hewlett Packard HP 9000/350
    OS&VER #: HP-UX, Version 4.0
Target Architecture ISA: Hewlett Packard HP64000 with Tasco MDC281 Emulator
    OS&VER #: TLDspk Single Program Kernel, Version 2.1.3

Implementor's Declaration

I, the undersigned, representing TLD Systems, Ltd., have implemented no
deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the
compiler listed in this declaration operating in the default mode. I declare
that TLD Systems, Ltd. is the owner of record of the Ada language compiler
listed above and, as such, is responsible for maintaining said compiler in
conformance to ANSI/MIL-STD-1815A. All certificates and registrations for
the Ada language compiler listed in this declaration shall be made only in
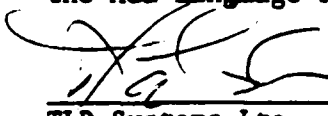the owner's corporate name.

_____   Date: __29 January 1990_____
TLD Systems, Ltd.
Terry L. Dunbar, President

Owner's Declaration

I, the undersigned, representing TLD Systems, Ltd., take full responsibility
for implementation and maintenance of the Ada compiler listed above, and
agree to the public disclosure of the final Validation Summary Report. I
further agree to continue to comply with the Ada trademark policy, as defined
by the Ada Joint Program Office. I declare that all of the Ada language
compilers listed, and their host/target performance are in compliance with
the Ada Language Standard ANSI/MIL-STD-1815A.

_____   Date: __29 January 1990_____
TLD Systems Ltd.
Terry L. Dunbar, President

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to
implementation-dependent pragmas, to certain machine-dependent conventions
as mentioned in Chapter 13 of the Ada Standard, and to certain allowed
restrictions on representation clauses. The implementation-dependent
characteristics of the TLD HP/MIL-STD-1750A Ada Compiler System, Version
2.1.3 compiler, as described in this Appendix, are provided by TLD Systems,
Ltd. Unless specifically noted otherwise, references in this Appendix are
to compiler documentation and not to this report. Implementation-specific
portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is

   ...

   type INTEGER is range -32768 .. 32767;
   type LONG_INTEGER is range -2147483648 .. 2147483647;

   type FLOAT is digits 6 range -1.0*2.0**127 .. 0.999999*2.0**127;
   type LONG_FLOAT is digits 9 range -1.0*2.0**127 .. 0.999999999*2.0**127;

   type DURATION is delta 2.0**(-14) range -86400.0 .. 86400.0;

   ...

end STANDARD;
```

The Ada language definition allows for certain machine_dependencies in a controlled manner. No machine-dependent syntax or semantic extensions or restrictions are allowed. The only allowed implementation-dependencies correspond to implementaton-dependent pragmas and attributes, certain machine-dependent conventions as mentioned in chapter 13, and certain allowed restrictions on representation clauses.

The full definition of the implementation-dependent characteristics of the TLD HP/MIL-STD-1750A Ada Compiler System is presented in this Appendix F.

Implementation-Dependent Pragmas

The TLD ACS supports the following implementation dependent pragmas.

Pragma Collect (type_name, attribute);

This pragma tells the compiler to collect all objects of specified type_name and subtypes of type_name into unmapped control sections. An "unmapped control section" is allocated a physical memory location not covered by a page register. Unmapped control sections are accessed from a device by DMA or by IBM GVSC extended instructions.

Pragma Control_Section (USECT,UNMAPPED,object_name(,object_name...));

This pragma specifies data objects that will be put into unmapped control sections. The first two parameters must be USECT and UNMAPPED and the remaining parameters are the names of Ada objects. An "unmapped control section" is allocated a physical memory location not covered by a page register. Unmapped control sections are accessed from a device by DMA or by IBM GVSC extended instructions.

Pragma Export (Language_name, Ada_entity_name, (String));

This pragma is a complement to Pragma Interface and instructs the compiler to make the entity named available for reference by a foreign language module. The language name identifies the language in which the foreign module is coded. The only foreign language presently supported is Assembly. Ada and JOVIAL are permitted and presently mean the same as Assembly but the semantics of their use are subject to redefinition by future releases of the compiler. If the optional third parameter is present, the string provides the name by which the entity may be referenced by the foreign module. The contents of this string must conform to the conventions for the indicated foreign language and the linker being used. No check is made by the compiler to insure that these conventions are obeyed.

Only objects having static allocation and subprograms are supported by pragma Export. If the Ada entity named is a subprogram, this pragma must be placed

within the declarative region of the named subprogram. If the name is that of an object, the pragma must be placed within the same declarative region and following the object declaration. It is the responsibility of the programmer to insure that the subprogram and object are elaborated before the reference is made.

```
pragma If (Compile_Time_Expression);
pragma Elsif (Compile_Time_Expression);
pragma Else;
pragma Endif;
```

These source directives may be used to enclose conditionally compiled source to enhance program portability and configuration adaptation. These directives may occur at the place that language defined pragmas, statements, or declarations may occur. Source code following these pragmas will be compiled or ignored similar to the semantics of the corresponding Ada statements depending upon whether the Compile_Time_Expression is true or false, respectively. The primary difference between these pragmas and the corresponding Ada statements are that the pragmas may enclose declarations and other pragmas.

```
Pragma Interrupt_Kind (Entry_Name, Entry_Type(, Duration));
```

This pragma must appear in the task specification and must appear after the Entry_Name is declared. Allowed Entry_Types are Ordinary, Timed, and Conditional. The optional parameter Duration is applicable only to timed entries and is the time to wait for an accept.

For an Ordinary entry, if the accept is not ready, the task is queued.

For a Timed entry, if the accept is not ready, the program waits for the period of time specified by the Duration. If the accept does not become ready in that period, the interrupt is ignored.

For a Conditional entry, if the accept is not ready, the interrupt is ignored.

```
Pragma Monitor;
```

Pragma Monitor can eliminate tasking context overhead. The pragma identifies Ada tasks that obey certain restrictions (listed below), allowing efficient invocaton by the compiler. With Pragma Monitor, a simple procedure call is used to invoke task entry.

The pragma only applies to tasks that have the following restrictions:

o Monitor tasks must only be declared in library level non-generic packages.

o Monitor tasks may contain data declarations only within the accept statement.

o A monitor task consists of an infinite loop containing one select statement.

o The "when condition" is not allowed in the select alternative of the select statement.

o The only selective wait alternative allowed in the select statement is the accept alternative.

o All executable statements of a monitor task must occur within an accept statement.

o Only one accept statement is allowed for each entry declared in the task specification.

If a task body violates restrictions placed on monitor tasks, it is identified as erroneous and the compilation fails.

pragma No_Default_Initialization;
pragma No_Default_Initialization (typename, (,typename ...));

The LRM requires initialization of certain data structures even though no explicit initialization is coded. For example, the LRM requires access_type objects to have an initial value of "NULL." The No_Default_Initialization pragma would prevent this default initialization.

In addition, initialization declared in a type statement is suppressed by this pragma.

The TLD implementation of packed records or records with representation clauses includes default initialization of filler bits, i.e., bits within the allocated size of a variant that are not associated with a record component for the variant. No_Default_Initialization prevents this default initialization.

No_Default_Initialization must be placed in the declaration region of the package, before any declarations that require elaboration code. The pragma remains in effect until the end of the compilation unit.

Pragma No_Elaboration;

Pragma No_Elaboration is used to prevent the generation of elaboration code for the containing scope. The pragma must be placed in the declaration region of the affected scope before any declaration that would otherwise

produce elaboration code.

Pragma No_Elaboration prevents otherwise unnecessary initialization of packages that are initialized by other non-Ada operations. Examples are ROM data and Real Time Kernel initialization. It is used to maintain the TLD Run Time Library (TLDrtl) and is not intended for general use.

Implementation-Dependent Attributes

None.

Representation Clause Restrictions

Pragma Pack is not supported.

Length clauses are not supported for 'SIZE except when applied to task types.

Length clauses are not supported for 'Storage_Size when applied to access types.

Length clauses are supported for 'Storage_Size when applied to a task type and denote the number of words of stack to be allocated to the task.

Length clauses are not supported for 'Small.

Enumeration representation clauses are supported for value ranges of Integer'First to Integer'Last.

Record representation clauses are supported to arrange record components within a record. Record components may not be specified to cross a word boundary unless they are arranged to encompass two or more whole words. A record component of type record that has itself been "rep specificationed" may only be allocated at bit 0. Bits are numbered from left to right with bit 0 indicating the sign bit.

The alignment clause is not supported.

Address clauses are supported for variable objects and designate the virtual address of the object. The TLD Ada Compiler System treats the address specification as a means to access objects allocated by other than Ada means and accordingly does not treat the clause as a request to allocate the object at the indicated address.

Address clauses are not supported for constant objects, packages, tasks, or task entries.

Implementation-Generated Names

The TLD Ada Compiler System defines no implementation dependent names for compiler generated components.

Address Clause Expressions

Address expression values and type Address represent a location in logical memory (in the program's current address state). For objects, the address specifies a location within the 64K word logical operand space. The 'Address attribute applied to a subprogram represents a 16 bit word address within the logical instruction space.

Unchecked Conversion Restrictions

None

I/O Package Characteristics

The following implementation-defined types are declared in Text_Io.

        subtype Count is integer range 0 .. 511;

        subtype Field is Integer range 0 .. 127;

Package Standard

The implementation-defined types of package Standard are:

        type Integer is   range -32_768 .. 32_767;
        type Long_Integer is   range -2_147_483_648 .. 2_147_483_647;
        type Float is digits 6 range -1.0*2.0**127 .. 0.999999*2.0**127;
        type Long_Float is digits 9 range -1.0*2.0**127 .. 0.999999999*2.0**127;
        type Duration is delta 2.0**(-14) range -86_400.0..86_400.0;

Other System Dependencies

LRM Chapter 1.

   None.

LRM Chapter 2.

Maximum source line length -- 120 characters.

Source line terminator -- Determined by the Editor used.

Maximum name length -- 120 characters.

External representation of name characters.

Maximum String literal -- 120 characters.

LRM Chapter 3.

LRM defined pragmas are recognized and processed as follows:

    Controlled -- Has no effect.

    Elaborate -- As described in the LRM.

    Inline -- Not presently supported.

    Interface -- Supported as a means of importing foreign language components into the Ada Program Library. May be applied either to a subprogram declaration as being specially implemented, -- read Interface as Import --, or to an object that has been declared elsewhere. Interface languages supported are System for producing a call obeying the standard calling conventions except that the BEX instruction is used to cause a software interrupt into the kernel supervisor mode; Assembly for calling assembly language routines; and Mil-Std-1750A for defining built in instruction procedures. An optional third parameter is used to define a name other than the name of the Ada subprogram for interfacing with the linker.

    List -- As defined in the LRM.

    Memory Size -- Has no effect.

    Optimize -- Has no effect. Optimization controlled by compiler command option.

    Pack -- Has no effect.

    Page -- As defined in the LRM.

Priority -- As defined in the LRM. Priority may range from 0 to 16366.
Default priority is 1.

Shared -- As defined in the LRM. May be applied to scalar objects only.

Storage Unit -- Has no effect.

Suppress -- As defined in the LRM for suppressing checks; all standard
checks may be suppressed individually as well as "Exception_Info" and
"All_Checks".   Suppression of Exception_Info eliminates data used to
provide symbolic debug information in the event of an unhandled
exception. The All_Checks selection eliminates all checks with a single
pragma.   In addition to the pragma, the compiler permits control of
check suppression by command line option without the necessity of source
changes.

System Name -- Has no effect.

Number declarations are not assigned addresses and their names are not
permitted as a prefix to the 'address attribute. (Clarification only).

Objects are allocated by the compiler to occupy one or more 16 bit 1750A
words. Only in the presence record representation clauses are objects
allocated to less than a word.

Except for access objects, uninitialized objects contain an undefined value.
An attempt to reference the value of an uninitialized object is not detected.

The maximum number of enumeration literals of all types is limited only by
available symbol table space.

The predefined integer types are:

Integer range -32_768 .. 32_767 and is implemented as a 1750A single
precision fixed point data.
Long_Integer range -2_147_483_648 .. 2_147_483_647 and implemented as
1750A double precision data.
Short_Integer is not supported.

System.Min_Int is -2_147_483_648.
System.Max_Int is 2_147_483_647.

The predefined real types are:

Float digits 6.
Long_Float digits 9.
Short_Float is not presently supported.

System.Max_Digits is presently 9 and is implemented as 1750A 48-bit
floating point data.

Fixed point is implemented as 1750A single and double precision data as is appropriate for the range and delta.

On the 1750A, index constraints as well as other address values such as access types are limited to an unsigned range of 0 .. 65_536 or a signed range of -32_768 .. 32_767.

The maximum array size is limited to the size of virtual memory -- 64K words.

The maximum String length is the same as for other arrays.

Access objects are implemented as an unsigned 16 bit 1750A integer. The access literal Null is implementated as one word of zero on the 1750A.

There is no limit on the number of dimensions of an array type. Array types are passed as parameters opposite unconstrained formal parameters using a 3 word dope vector illustrated below:

```
-----------------------------------------
|    Word address of first element       |
|    Low bound value of first dimension   |
|    Upper bound value of first dimension |
-----------------------------------------
```

Additional dimension bounds follow immediately for arrays with more than one dimension.

LRM Chapter 4.

Machine_Overflows is True for the 1750A.

Pragma Controlled has no effect for the TLD VAX/1750A Compiler since garbage collection is never performed.

LRM Chapter 5.

The maximum number of statements in an Ada source program is undefined and limited only by Symbol Table space.

Case statements unless they are quite sparse, are allocated as indexed jump vectors and are, therefore, quite fast.

Loop statements with a for implementation scheme are implemented most efficiently on the 1750A if the range is in reverse and down to zero.

Data declared in block statements on the 1750A is elaborated as part of its containing scope.

LRM Chapter 6.

Arrays, records and task types are passed on the 1750A by reference.

Pragma Inline is not presently supported for subprograms.

LRM Chapter 7.

Package elaboration is performed dynamically permitting a warm restart without the necessity to reload the program.

LRM Chapter 8.

LRM Chapter 9.

Task objects are implemented as access types pointing to a Task Information Block (TIB).

Type Time in package Calendar is declared as a record containing two double precision integer values: the date in days and the real time clock.

Pragma Priority is supported with a value of 1 to 16366.

Pragma Shared is supported for scalar objects.

LRM Chapter 10.

Multiple Ada Program Libraries are supported with each library containing an optional ancester library. The predefined packages are contained in the TLD standard library, ADA.LIB.

LRM Chapter 11.

Exceptions are implemented by the TLD Ada Compiler System to take advantage of the normal policy in embedded computer system design to reserve 50% of the duty cycle. By executing a small number of instructions in the prologue of a procedure or block containing an exception handler, a branch may be taken, at the occurrence of an exception, directly to a handler rather than performing the time consuming code of unwinding procedure calls and stack frames. The philosophy taken is that an exception signals an exceptional condition, perhaps a serious one involving recovery or reconfiguration, and that quick response in this situation is more important and worth the small throughput tradeoff in a real time environment.

LRM Chapter 12.

A single generic instance is generated for a generic body. Generic specifications and bodies need not be compiled together nor need a body be compiled prior to the compilation of an instantiation. Because of the single expansion, this implementation of generics tend to be more favorable on the 1750A because of the usual space savings achieved. To achieve this tradeoff, the instantiations must by nature be more general and are, therefore, somewhat less efficient timewise.

LRM Chapter 13.

Representation clause support and restrictions are defined above.

A comprehensive Machine_Code package is provided and supported.

Unchecked_Deallocation and Unchecked_Conversion are supported.

The implementation dependent attributes are all supported except 'Storage_Size for an access type.

LRM Chapter 14.

File I/O operations are not supported for the 1750A. Text_Io and Low_Level_Io are supported.

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

Name and Meaning	Value

$ACC_SIZE
    An integer literal whose value
    is the number of bits sufficient
    to hold any value of an access
    type.
	16

$BIG_ID1
    An identifier the size of the
    maximum input line length which
    is identical to $BIG_ID2 except
    for the last character.
	(1..119 => 'A', 120 => '1')

$BIG_ID2
    An identifier the size of the
    maximum input line length which
    is identical to $BIG_ID1 except
    for the last character.
	(1..119 => 'A', 120 => '2')

$BIG_ID3
    An identifier the size of the
    maximum input line length which
    is identical to $BIG_ID4 except
    for a character near the middle.
	(1..80 => 'A', 81 => '3',
	 82..120 => 'A')

TEST PARAMETERS

Name and Meaning	Value
$BIG_ID4 An identifier the size of the maximum input line length which is identical to $BIG_ID3 except for a character near the middle.	(1..80 => 'A', 81 => '4', 82..120 => 'A')
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	(1..117 => '0', 118..120 => "298")
$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	(1..115 => '0', 116..120 => "690.0")
$BIG_STRING1 A string literal which when catenated with $BIG_STRING2 yields the image of $BIG_ID1.	(1 => '"', 2..61 => 'A', 62 => '"')
$BIG_STRING2 A string literal which when catenated to the end of $BIG_STRING1 yields the image of $BIG_ID1.	(1 => '"', 2..60 => 'A', 61 => '1', 62 => '"')
$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	(1..100 => ' ')
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	511
$DEFAULT_MEM_SIZE An integer literal whose value is SYSTEM.MEMORY_SIZE.	65536
$DEFAULT_STOR_UNIT An integer literal whose value is SYSTEM.STORAGE_UNIT.	16

Name and Meaning	Value
$DEFAULT_SYS_NAME The value of the constant SYSTEM.SYSTEM_NAME.	AF1750
$DELTA_DOC A real literal whose value is SYSTEM.FINE_DELTA.	2.0**-31
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	127
$FIXED_NAME The name of a predefined fixed-point type other than DURATION.	NO_SUCH_TYPE
$FLOAT_NAME The name of a predefined floating-point type other than FLOAT, SHORT_FLOAT, or LONG_FLOAT.	NO_SUCH_TYPE
$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	90000.0
$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	131073.0
$HIGH_PRIORITY An integer literal whose value is the upper bound of the range for the subtype SYSTEM.PRIORITY.	16#3FEE#
$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	"BADCHAR@.!"
$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	"THISFILENAMEWOULDBEPERFECTLYLEGALIF"
$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-32768

TEST PARAMETERS

Name and Meaning	Value
$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	32767
$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	32768
$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-90000.0
$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-131073.0
$LOW_PRIORITY An integer literal whose value is the lower bound of the range for the subtype SYSTEM.PRIORITY.	1
$MANTISSA_DOC An integer literal whose value is SYSTEM.MAX_MANTISSA.	31
$MAX_DIGITS Maximum digits supported for floating-point types.	9
$MAX_IN_LEN Maximum input line length permitted by the implementation.	120
$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	2_147_483_647
$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2_147_483_648
$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be $MAX_IN_LEN long.	(1..2 => "2:", 3..117 => '0', 118..120 => "11:")

Name and Meaning	Value
$MAX_LEN_REAL_BASED_LITERAL A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be $MAX_IN_LEN long.	(1..3 => "16:", 4..116 => '0', 117..120 => "F.E:")
$MAX_STRING_LITERAL A string literal of size $MAX_IN_LEN, including the quote characters.	(1 => '"', 2..119 => 'A', 120 => '"')
$MIN_INT A universal integer literal whose value is SYSTEM.MIN_INT.	-2_147_483_648
$MIN_TASK_SIZE An integer literal whose value is the number of bits required to hold a task object which has no entries, no declarations, and "NULL;" as the only statement in its body.	16
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	AF1750
$NAME_LIST A list of enumeration literals in the type SYSTEM.NAME, separated by commas.	AF1750
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFE
$NEW_MEM_SIZE An integer literal whose value is a permitted argument for pragma MEMORY_SIZE, other than $DEFAULT_MEM_SIZE. If there is no other value, then use $DEFAULT_MEM_SIZE.	65536

TEST PARAMETERS

Name and Meaning	Value
$NEW_STOR_UNIT An integer literal whose value is a permitted argument for pragma STORAGE_UNIT, other than $DEFAULT_STOR_UNIT. If there is no other permitted value, then use value of SYSTEM.STORAGE_UNIT.	16
$NEW_SYS_NAME A value of the type SYSTEM.NAME, other than $DEFAULT_SYS_NAME. If there is only one value of that type, then use that value.	AF1750
$TASK_SIZE An integer literal whose value is the number of bits required to hold a task object which has a single entry with one 'IN OUT' parameter.	16
$TICK A real literal whose value is SYSTEM.TICK.	1.0/10000.0

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the
Ada Standard. The following 44 tests had been withdrawn at the time of
validation testing for the reasons indicated. A reference of the form
AI-ddddd is to an Ada Commentary.

a. E28005C: This test expects that the string "-- TOP OF PAGE. --63" of
   line 204 will appear at the top of the listing page due to a pragma
   PAGE in line 203; but line 203 contains text that follows the pragma,
   and it is this text that must appear at the top of the page.

b. A39005G: This test unreasonably expects a component clause to pack an
   array component into a minimum size (line 30).

c. B97102E: This test contains an unintended illegality: a select
   statement contains a null statement at the place of a selective wait
   alternative (line 31).

d. C97116A: This test contains race conditions, and it assumes that
   guards are evaluated indivisibly. A conforming implementation may use
   interleaved execution in such a way that the evaluation of the guards
   at lines 50 & 54 and the execution of task CHANGING_OF_THE_GUARD
   results in a call to REPORT.FAILED at one of lines 52 or 56.

e. BC3009B: This test wrongly expects that circular instantiations will
   be detected in several compilation units even though none of the units
   is illegal with respect to the units it depends on; by AI-00256, the
   illegality need not be detected until execution is attempted (line
   95).

f. CD2A62D: This test wrongly requires that an array object's size be no
   greater than 10 although its subtype's size was specified to be 40
   (line 137).

g. CD2A63A..D, CD2A66A..D, CD2A73A..D, and CD2A76A..D (16 tests): These
    tests wrongly attempt to check the size of objects of a derived type
    (for which a 'SIZE length clause is given) by passing them to a
    derived subprogram (which implicitly converts them to the parent type
    (Ada standard 3.4:14)). Additionally, they use the 'SIZE length
    clause and attribute, whose interpretation is considered problematic
    by the WG9 ARG.

h. CD2A81G, CD2A83G, CD2A84M..N, and CD50110 (5 tests): These tests
    assume that dependent tasks will terminate while the main program
    executes a loop that simply tests for task termination; this is not
    the case, and the main program may loop indefinitely (lines 74, 85,
    86, 96, and 58, respectively).

i. CD2B15C and CD7205C: These tests expect that a 'STORAGE_SIZE length
    clause provides precise control over the number of designated objects
    in a collection; the Ada standard 13.2:15 allows that such control
    must not be expected.

j. CD2D11B: This test gives a SMALL representation clause for a derived
    fixed-point type (at line 30) that defines a set of model numbers that
    are not necessarily represented in the parent type; by Commentary
    AI-00099, all model numbers of a derived fixed-point type must be
    representable values of the parent type.

k. CD5007B: This test wrongly expects an implicitly declared subprogram
    to be at the address that is specified for an unrelated subprogram
    (line 303).

l. ED7004B, ED7005C..D, and ED7006C..D (5 tests): These tests check
    various aspects of the use of the three SYSTEM pragmas; the AVO
    withdraws these tests as being inappropriate for validation.

m. CD7105A: This test requires that successive calls to CALENDAR.CLOCK
    change by at least SYSTEM.TICK; however, by Commentary AI-00201, it is
    only the expected frequency of change that must be at least
    SYSTEM.TICK--particular instances of change may be less (line 29).

n. CD7203B and CD7204B: These tests use the 'SIZE length clause and
    attribute, whose interpretation is considered problematic by the WG9
    ARG.

o. CD7205D: This test checks an invalid test objective: it treats the
    specification of storage to be reserved for a task's activation as
    though it were like the specification of storage for a collection.

p. CE2107I: This test requires that objects of two similar scalar types be distinguished when read from a file--DATA_ERROR is expected to be raised by an attempt to read one object as of the other type. However, it is not clear exactly how the Ada standard 14.2.4:4 is to be interpreted; thus, this test objective is not considered valid (line 90).

q. CE3111C: This test requires certain behavior, when two files are associated with the same external file, that is not required by the Ada standard.

r. CE3301A: This test contains several calls to END_OF_LINE and END_OF_PAGE that have no parameter: these calls were intended to specify a file, not to refer to STANDARD_INPUT (lines 103, 107, 118, 132, and 136).

s. CE3411B: This test requires that a text file's column number be set to COUNT'LAST in order to check that LAYOUT_ERROR is raised by a subsequent PUT operation. But the former operation will generally raise an exception due to a lack of available disk space, and the test would thus encumber validation testing.

APPENDIX E

COMPILER OPTIONS AS SUPPLIED BY TLD SYSTEMS, LTD.

Compiler: TLD HP/MIL–STD–1750A Ada Compiler System, Version 2.1.3

ACVC Version: 1.10

In all of the above examples, the input file test.ada is compiled
generating a relocatable object file named test.obj, a listing file
named test.lst containing a summary of the option switches, a source
image and generated assembly listing and a cross reference listing. The   ?
Ada source code is interspersed with the generated assembly code in the
listing file. No constraint checking code is generated.   All address
computations are performed in 1750A single precision fixed point
arithmetic. A system cross reference file, test.srf is also generated.   ?

3.2 Compiler Recognized Names

3.3 Compiler Option Switches

Compiler option switches provide control over various processing and
output features of the compiler. These features include several
varieties of listing output, the level and kinds of optimizations
desired, the choice of target computer, and the operation of the
compiler in a syntax checking mode only.

Keywords are used for selecting various compiler   options.   The
complement keyword, if it exists, is used to disable a compiler option
and is formed by prefixing the switch keyword with "no_".

Switches may be abbreviated to the number of characters required to
uniquely identify the switch.   For example, the switch "elaborator"
(explained in the list, below) may be uniquely identified by the
abbreviation "el." Of course, the entire name may be used.   If an
option is not specified by the user, a default setting is assumed.   All
specified compiler options apply to a single invocation of the compiler.

The default setting of a switch and its meaning are defined in the table
below.   The meaning of the complement form can be derived by
complementing the meaning of the switch. For some switches, the
complement meaning is not obvious; these complement switch keywords are
listed separately.

In the description of the switches, the target dependent name <target>
is used.   The value of this symbol is determined by the value of the
target switch.

Compiler generated file specifications generally conform to host
conventions. For example:

The generated file name is the source filename appended with the default
file suffix.   If the user specifies an output file name, no default
suffix is provided.

Switch Name   Meaning
....

16bador
32bador -- default

> The 32BADDR option causes address computations to be performed
> using 1750A double precision fixed point data words.   If
> 16BADDR is selected, address computations will be performed
> using single precision fixed point data words ignoring the
> possibility of a 1750A Fixed Point Overflow Interrupt due to
> computation of an address greater than 7FFF hex.   Applicable
> to 1750A target only.

checks -- default
checks(-check_identifier)
checks(-check_identifier(,...))
no_checks
no_checks(-check_identifier)
no_checks(-check_identifier(,...))

> The checks
>
> switch without check_identifiers enables all run time checks.
> If one or more check_identifiers are specified, the specified
> run time checks are enabled; the status of run time checks
> associated with unmentioned check_identifiers is unchanged.
>
> The no_checks switch omits all run time checks.   If one or
> more check_identifiers are specified, the specified run time
> checks are omitted; the status of run time checks associated
> with unmentioned check_identifiers is unchanged. With one
> exception, no_checks pragma Suppress(all_checks) in the source
> code.   The exception is associated with the TLD defined
> check_identifier exception_info.
>
> Check_identifiers are listed below and are described in the
> LRM, Section 11.7 except exception_info.

access_check	discriminant_check	division_check
elaboration_check	exception_info	index_check
length_check	overflow_check	range_check
storage_check		

The TLD defined check_identifier exception_info is explained below.

TLDada generates a string in the relocatable object code that is the full path name of the file being compiled. TLDada also generates extra instructions in the relocatable code that are used by the Run Time System to inform the programmer where an unhandled exception is raised. The no_checks-exception_info switch will suppress both the string and extra instructions, whereas the pragma will only suppress the extra instructions.

codegen -- default
no_codegen

The no_codegen switch causes the compiler to check syntax and update the Ada Program Library, but no code is generated.

cpl-n
cpl-110 -- default

The cpl switch specifies the number of characters per line in the listing file. The cpl value can range from 80 to 132.

crossref
no_crossref -- default

The crossref switch generates a cross-reference listing which includes referenced names only. The cross-reference listing is included in the listing file; therefore the list switch must be selected or crossref has no effect.

cseg -- default
no_cseg

The cseg switch requests that constants be allocated in a control section of their own (1750A target only).

debug -- default
no_debug

The debug switch selects the production of symbolic debug tables in the relocatable object file that permit access to Ada source program names, attributes, and source line numbers at run time.
Alternate abbreviation: dbg, no_dbg

delassign -- default
no_delassign

>The delassign switch optimizes code by deleting redundant assignments. Note: Use of this switch can cause erroneous source programs to execute with unexpected results if references to access objects are made without regard to the interference semantics of Ada.

elaborator

>The elaborator switch informs TLDada that the specified source-file-spec is the name of the Ada Program Library unit that is to be used as the main subprogram.

errlevel-n
errlevel-3 -- default

>The errlevel switch assigns a minimum error level value which will terminate compilation at the completion of the compiler phase encountering an error equal to or greater than the indicated level. The value can range from 0 to 3.

gopt -- default
gopt(optswitch)
gopt(optswitch(,...))
no_gopt -- temporarily disabled

>The gopt switch cannot be disabled in the currently released Ada Compiler. i The gopt switch enables global optimization and selects which optimizations are to be performed by the global optimizer. Section 3.8 of the Ada Compiler Reference Manual provides further information on optimization. Selection of the gopt switch enables the following default optswitch options as indicated unless overridden by an explicit optswitch selection.

>Optswitch Name   Meaning

>cse -- default
>no_cse
>>Common subexpression elimination.

>fold -- default
>no_fold
>>Folding of constants, scalars, and expressions.

>adel -- default
>no_adel

Dead assignment deletion.

hoist -- default
no_hoist
Code hoisting.

icm -- default
no_icm
Invariant code motion out of loops.

stred -- default
no_stred
Strength reduction.

trep -- default
no_trep
Test replacement.

rgal -- default
no_rgal
Register allocation.

extc
no_extc -- default
External calls may exit with abort

safe -- default
no_safe
Code hoisting and invariant code motion are performed only if no new computations are executed.

indent=n
indent=3 -- default
no_indent

The indent switch produces a formatted (indented) source listing. This switch assigns a value to the number of columns used in indentation; the value n can range from 0 to 15.

info -- default
no_info

The info switch outputs all diagnostic messages regardless of level. The no_info switch suppresses the output of information-level diagnostic messages.

intsl   ?
no_intsl -- default   ?
                                                                         ?
      The intsl switch intersperses lines of source code with the   ?
assembly code it generates in the macro listing. This switch   ?
is valid only if the list, source and macro switches are   ?
selected.   ?
                                                                         ?

library Ada-program-library-file-spec
library <target>.lib -- default

      The library switch specifies the file to be used for Ada
Program Library. The default value of <target> in the Ada
Program Library file spec is derived from the target switch.

list(-listing-file-spec)
no_list -- default

      The LIST switch generates a listing file. The default file
suffix is lis. The listing-file-spec can be optionally
specified. The listing will be directed to the user's
terminal if the listing-file-spec is /tty.

listcopy   ?
no_listcopy -- default   ?
                                                                         ?
      The listcopy switch lists the source lines of files included   ?
via !copy directives.   ?
Alternate abbreviations: lc, no_lc   ?

log
no_log -- default

      The log switch causes the Compiler to write to stdout the
source file spec of the file being compiled.

lpp-n
lpp-40 -- default

      The lpp switch assigns a value to the number of lines per page
for listing. The value can range from 10 to 99.
ADE Related Switch: /lpp

macro
no_macro -- default

> The macro switch produces an assembly like object code listing
> appended to the source listing file. The list switch must be
> enabled or this switch has no effect.
> ADE Related Switch: /ASSEMBLY

maint
no_maint -- default

> The maint switch is used only for compiler maintenance and
> debugging.

makelib(-parent-APL-spec)
no_makelib=default_RTS_spec -- default

> The makelib switch creates a new Ada Program Library (APL)
> file. This switch should be used with caution because it
> creates a new APL file in the default directory even if
> another APL file of the same name existed.

> The new APL file is created in the current working directory
> with the name <target>.lib unless the library switch is used.

> If makelib is used without a parent, a new library is created
> with the default RTS specification. This specification is
> derived from the name tld_lib_<target>. See the target
> dependent compiler sections for further explanations of this
> name.

maxerrors=n
maxerrors=500 -- default

> The maxerrors switch assigns a value limit to the number of
> errors forcing job termination. Once this value is exceeded,
> the compilation is terminated. Information-level diagnostic
> messages are not included in the count of errors forcing
> termination. The specified value's range is from 0 to 500.

model=model
model=standard -- default
      1750A target

> TLDada produces code for any implementation that conforms to
> MIL-STD-1750A. The model switch informs TLDada that a special
> model of the MIL-STD-1750A is the target computer. MODEL also
> is used to inform the compiler of 1750A implementations with
> special Built-In-Functions (BIFs). Models and the special
> features that characterize them are listed below.

> mdc281

>> This switch allows the compiler to make use of the
>> standard mdc281 BIF that produces a 64-bit integer
>> multiply result. In addition, the mdc281 chip set
>> has a known bug associated with FNEG and FABS in a
>> boundary condition. Use of this switch results in
>> generated code that avoids the problem, although the
>> generated code is less efficient.

> standard

>> All other implementations of MIL-STD-1750A are
>> supported by the default value of the model switch.

object(object-file-spec)
object -- default
no_object

> The object switch produces a relocatable object file. The
> default file suffix is ".obj".

opt -- default
no_opt

> The opt switch enables regional optimization on the compiled
> code.

parm
no_parm -- default

> The parm switch causes all option switches governing the
> compilation, including the defaulted option switches, to be
> included in the listing file. The list option must also be
> selected or parm will have no effect. User specified switches
> are preceded in the listing file by a leading asterisk (*).

pcross
no_pcross -- default

  The pcross switch selects a procedure call cross reference listing.

phase -- default
no_phase

  The no_phase switch suppresses displaying phase times during compilation.

reformat(reformat-file-spec)
no_reformat -- default

  The reformat switch causes TLDada to reformat the source listing in the listing file and, if a reformat-file-spec is present, to generate a reformatted source file. The default suffix of the new source file is ".rfm."

skip ((letter...))
no_skip -- default

  The skip switch permits the selective skipping of source code enclosed within !begin-!end directives. This option functions as if a !skip directive for each of the corresponding letter(s) had occurred at the beginning of the source program and if no letter is specified, exactly as if a !skip directive without any letter had occurred as the start of the source program, i.e., all !begin-!end enclosed source will be skipped. This option permits conditional compilation without changing the source file.

source -- default
no_source

  The source switch causes the input source program to be included in the listing file. Diagnostic messages, unless otherwise suppressed, are always included in the listing file.

sti (=sti-file-spec)
no_sti -- default

  The sti switch generates a software tools interface file. The sti-file-spec can be completely or partially specified. If only the sti file name is specified, the default sti file type, ".sti", is used to form the sti-file-spec. If only the file type is specified, the file name of the input-file-spec is used to form the sti-file-spec. If no sti-file-spec is specified, the sti file name is formed from the file name of the input-file-spec and the default sti file type, ".sti".

syntax_only
no_syntax only -- default

> The syntax_only switch performs syntax checking on the source program; no object file is produced and the macro switch is ignored. The Ada Program Library is not updated.

sysref(srf-file-spec)
no_sysref -- default

> The sysref switch generates an external name cross-reference file. This file consists of all external names referenced in the source program with the modules which refer to that external name. The default file suffix is SRF.

table_picture
no_table_picture -- default

> This switch selects a listing containing a graphical representation of the layout of table-items within table entries declared in the program.
> Alternate abreviation: tpic.

target 1750A -- default
target HAWK
target 68020
target HP300
target=MV

> The target switch selects the target computer for which code is to be generated for this compilation. "1750A" selects the MIL-STD-1750A Instruction Set Architecture, Notice A. "HAWK" selects the Rolm HAWK architecture running under either AOS/VS or ARTS/32. "68020" selects the Motorola 68020 architecture. "MV" selects the Eclipse MV architecture operating under AOS/VS. "HP300" selects the HP 9000 Series 300 Series architecture operating under HPUX. The 1750A is the only target currently released.